ABDULLAH GÜL
ÜNİVERSİTESİ

SAPA CAPSULE PROJECT
ELECTRONIC STETHOSCOPE

PROJECT TEAM

FATMA NUR ÖKSÜM
ABDULLAH RIHAWI
HUSEIN MOHAMED FAARAH

Table of Contents

Chapter 1

Introduction

A timeless representation of medical care and diagnostic proficiency is the stethoscope, a basic medical tool. The stethoscope, which was created in the early 1800s by French physician René Laennec, has grown to be a vital instrument for medical practitioners worldwide. A chest piece, tubing, and headphones make up the device's cleverly basic design, which enables practitioners to listen to internal noises including pulse, lung function, and gastrointestinal activities.

Figure 1. 1 Classic Sthethoscope

By expertly utilizing the stethoscope, medical professionals may get invaluable audio data about a patient's physiological condition, which aids in the diagnosis process. The stethoscope, an icon of the doctor-patient interaction, transcends its utilitarian purpose to represent the essence of understanding and healing. It symbolizes the meeting point of medical knowledge and compassionate care.

1.1 Objective

This capsule project's primary goal is to build an electronic stethoscope system that can detect acoustic sounds.

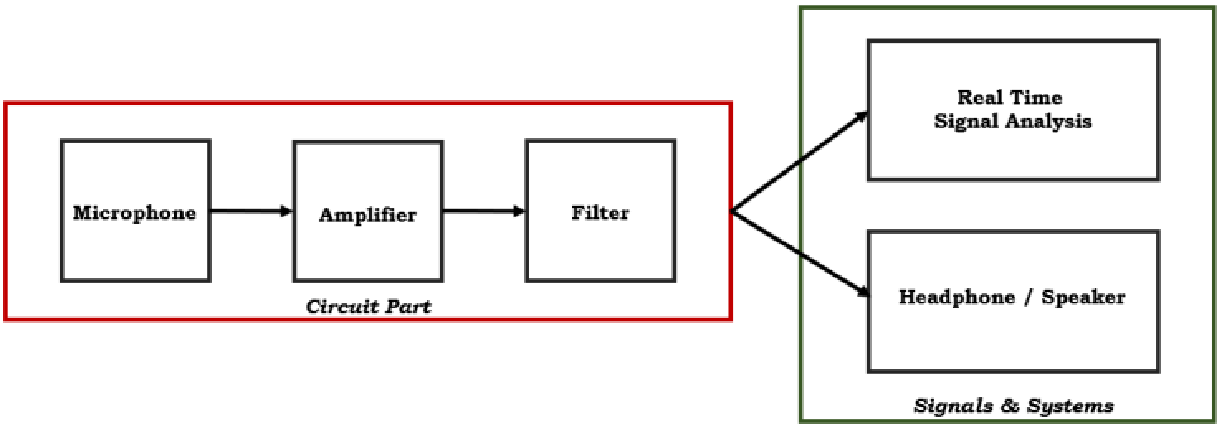


Figure 1.2 Project Flow Chart

It captures sound waves originating from different body areas and transforms them into a digital signal. To transfer these electronic signals to or from a personal computer, they can also be processed and digitalized with laptop. Both software and hardware will be included in this system.

The goal of this project is to create a digital stethoscope that can assess lung function, heart rate, and other parameters. sounds of an organ. The sound waves are first filtered and amplified by the gadget before being played back through speakers.

The gadget also transmits digital data to a computer for further uses, such as real-time signal monitoring.

1.2 Background

LOW PASS FILTERS

An essential part of an electrical circuit, a low-pass filter attenuates or blocks higher-frequency components while allowing signals with frequencies below a predetermined cutoff threshold to flow through. In essence, it reduces or filters the amplitudes of higher-frequency signals while allowing low-frequency information to flow through unaltered.

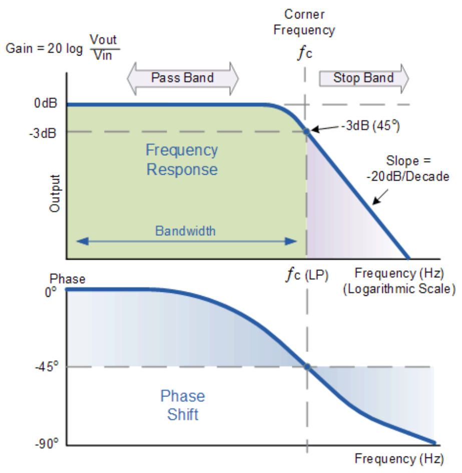


Figure 1.2.1 Low Pass Bode Plot

This kind of filter is widely used in many different applications, such as electronic signal processing, communication devices, and audio systems. The basic idea behind low-pass filters is that the amplitude of the signal gradually decreases as the frequency increases.

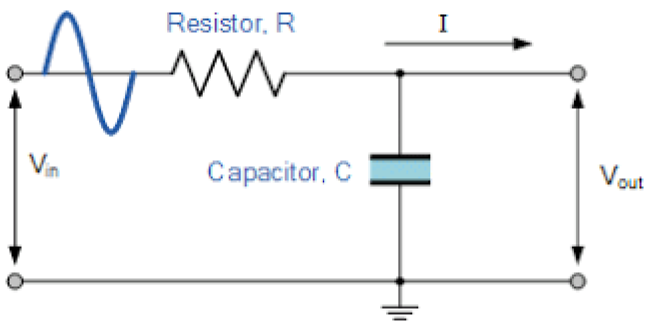


Figure 1.2.2 Low Pass Filter

Resulting in a seamless transition between the stopband and the passband. Low-pass filters are an essential instrument for shaping and regulating electronic signals because they remove high-frequency noise, improve signal clarity, and guarantee that only the pertinent frequency components reach a defined output.

HIGH PASS FILTERS

An essential part of electrical circuits that permit signals above a given cutoff point to go through while attenuating or blocking lower-frequency components is a high-pass filter. Its main purpose is to transmit higher-frequency signals while suppressing lower-frequency ones, which is the reverse of a low-pass filter.

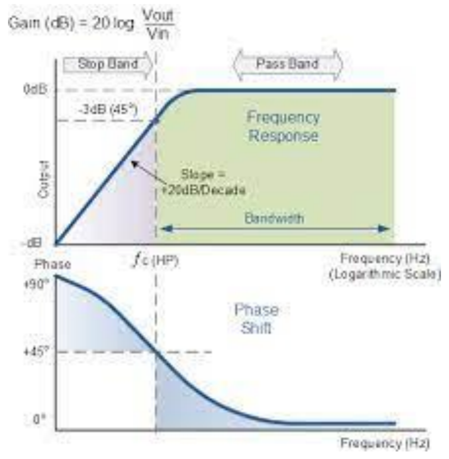
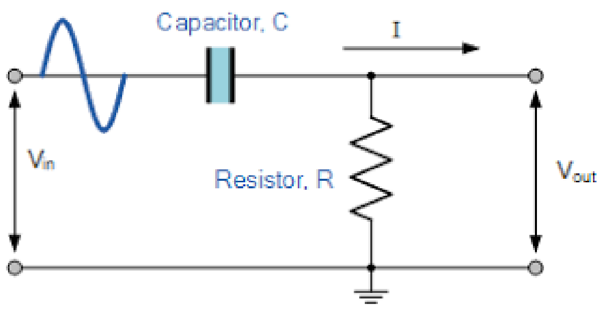


Figure 1.2.3 High Pass Bode Plot

High-pass filters are often used in many different applications, including signal processing, audio systems, and telecommunications. These filters assist in the removal of undesirable noise, improve the clarity of particular signal ranges



And adjust the frequency response of a system to suit desired requirements by selectively allowing the flow of higher-frequency information and restricting lower-frequency components. In order to provide a smooth and regulated response when frequencies rise over the designated threshold, high-pass filters are designed with a gentle transition between the passband and the stopband.

Figure 1.2.4 High Pass Filter

AMPLIFIERS

In the world of electronics, amplifiers are essential electronic devices that boost the intensity or amplitude of an electrical signal. These devices, which work on the fundamental concept of signal amplification, take a weak input signal and convert it into a stronger output signal, which is often expressed as voltage, current, or power.

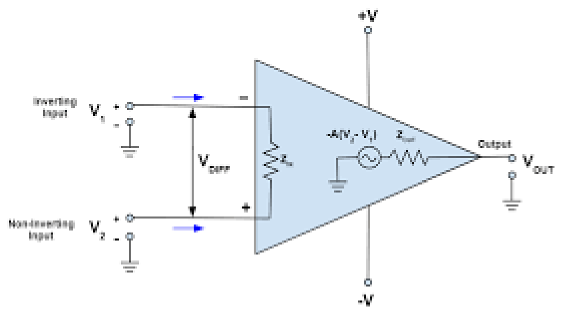


Figure 1.2.5 Op-amp simulation

Amplifiers are widely used in radio frequency (RF) communication, telecommunications, audio systems, and different electrical circuits. They are available in several varieties, each with a distinct function, including transistor amplifiers, power amplifiers, and operational amplifiers (op-amps). Amplifiers are essential to the functioning and performance of many common devices because they help overcome signal losses over transmission lengths, increase signal-to-noise ratios, and improve the fidelity and intensity of signals in electronic systems.

Figure 1.2.5 An op-amp

Chapter 2
Designing Circuit

2.1 MICROPHONE &AMPLIFIER CIRCUIT

Microphone

Electret microphones are a type of condenser microphone. They use a permanently charged material (the electret) as a polarizing mat erial instead of requiring an external power source for polarization

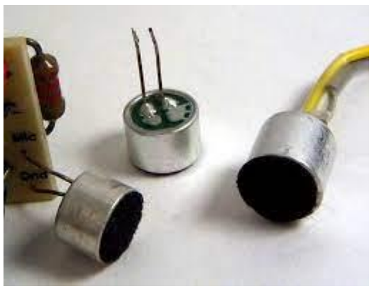


Figure 2.1.1 An Electret Microphone

Properties:

Electret microphones are generally sensitive and can capture a wide range of frequencies. These microphones have relatively low power requirements due to the electret's ability to hold a permanent charge.

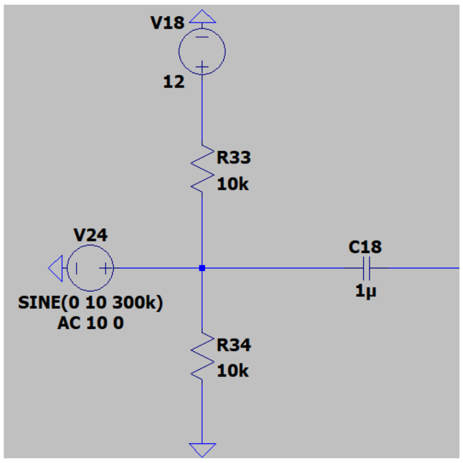


Figure 2.1.2 Voltage divider simulation

Voltage divider

In an electret microphone circuit, a voltage divider (also known as biasing or bias resistor network) is often used to establish a stable operating point or bias voltage for the microphone.

Noise Reduction Capacitor

Capacitors are used to reduce noise in electronic circuits.

Amplifier Circuit

In a stethoscope circuit, an amplifier plays a crucial role in magnifying the weak electrical signals generated by the microphone or transducer in response to body sounds (such as heartbeat or lung sounds) We used LM358 op-amp with a gain of 10k.

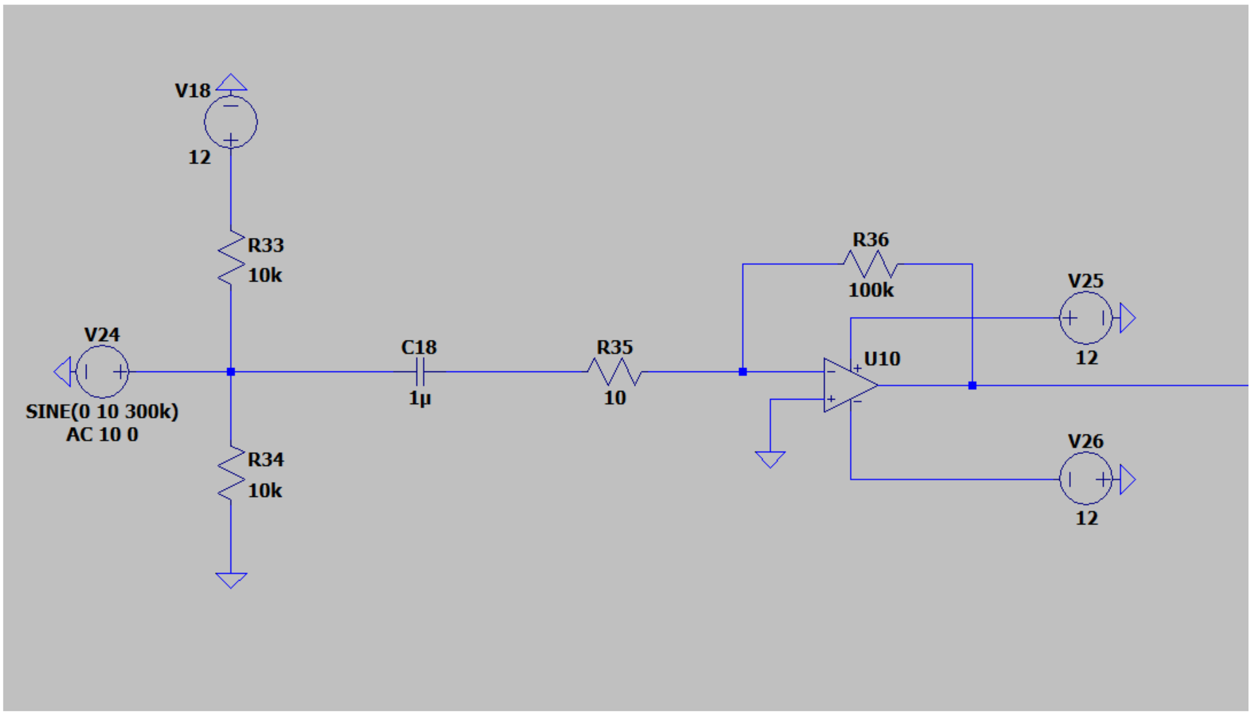


Figure 2.1.3 Amplifier Circuit Simulation

2.2 FILTER CIRCUIT

2.2.1 Fourth Order Low pass Butterworth Filter (for heart)

Heart sounds, including the first heart sound (S1) and second heart sound (S2), are typically recorded between 20 and 200 hertz (Hz). S1 quality decreased with closure of the tricuspid and mitral valves, usually at 20–30 Hz. Conversely, S2, which is linked to the closure of the pulmonary and aortic valves, has a somewhat higher frequency, in the 50–60 Hz range. These frequency ranges may change according on a person's age, heart rate, and other medical conditions. Heart sound frequency analysis is useful for improving overall heart health and making diagnostic diagnoses in clinical ranges.

Butterworth Filters

An electrical filter type known as a Butterworth filter has a frequency response that is defined by a maximum flat passband and a progressive stopband drop. These filters, which bear the name of British engineer Stephen Butterworth, are used for situations where maintaining signal amplitude is essential due to their uniform and steady amplitude response over the entire passband. As members of the infinite impulse response (IIR) filter family, butterworth filters include feedback into their designs. The filter order, a parameter that controls the transition's sharpness, defines the filter's design. Steeper roll-offs are the result of higher order Butterworth filters, however they can also introduce phase distortions. These filters are used in a variety of industries, including as communications, electronics, and signal processing, where precise and dependable filtering depends on a consistent frequency response.

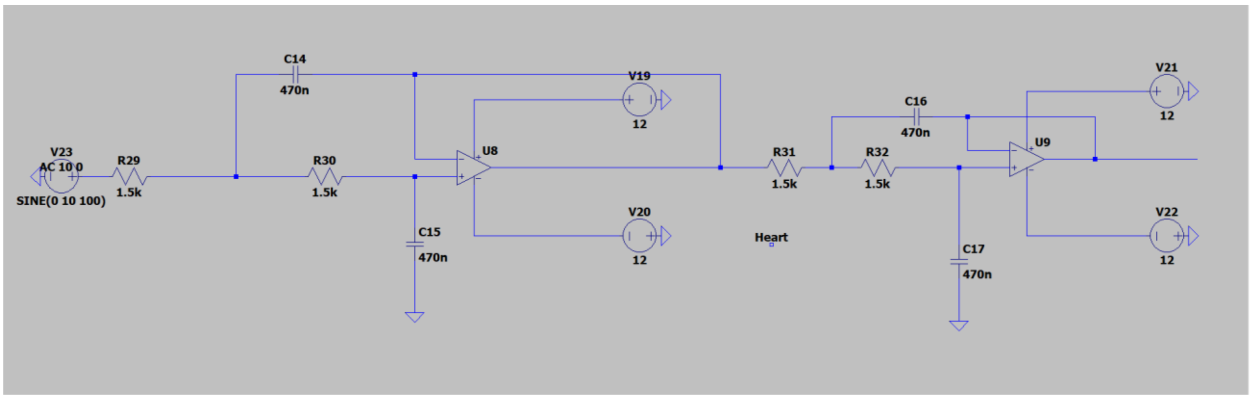


Figure 2.2.2 Fourth Order Low pass Butterworth Filter (for heart) Simulation

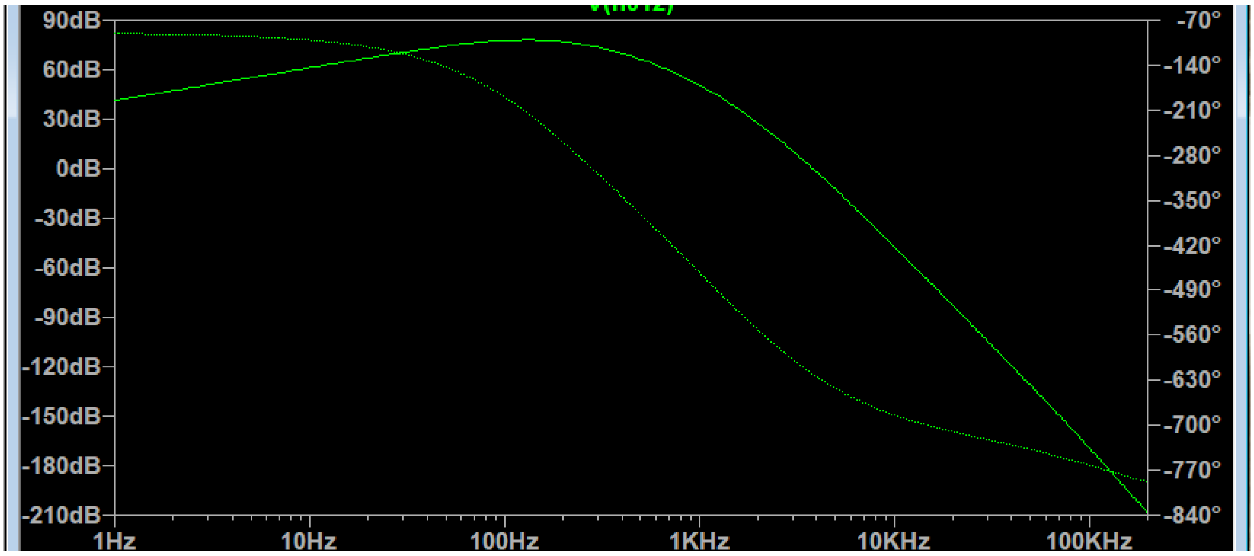


Figure 2.2.3 Fourth Order Low pass Butterworth Filter (for heart) Simulation Result

2.2.2. Second order Bandpass Butterworth Filter (for lungs)

Lung sounds assessed via auscultation are included in a series that provides valuable information about respiratory performance. Normal breath sounds generally fall into two main types: bronchial and vesicular. Branch sounds expanded by air moving through larger airways exhibit a higher frequency range, typically between 250 and 1000 Hertz (Hz). Vesicular features representing air movement in the small airways and alveoli have a lower frequency range; usually around 100 to 400 Hz. It turns out that respiratory rates are often observed in adventitious lungs, such as wheezing or crackling. Wheezing from the reflection of the airways has the feature of having a frequency between 400 and 1000 Hz. With the opening of the collapsed alveoli, crackles can vary between 200 and 2000 Hz. Analyzing the frequency components of lung sounds helps healthcare professionals diagnose respiratory diseases and record patients' lung diseases.

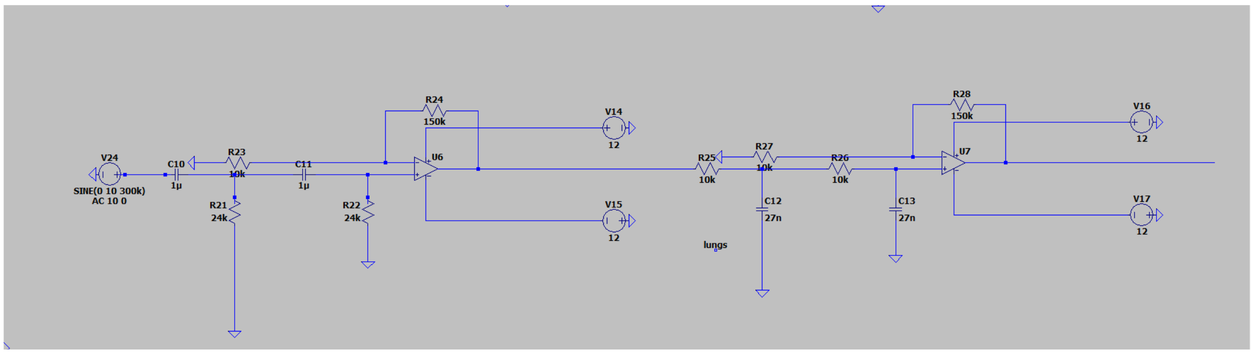


Figure 2.2.2 Second order Bandpass Butterworth Filter (for lungs)

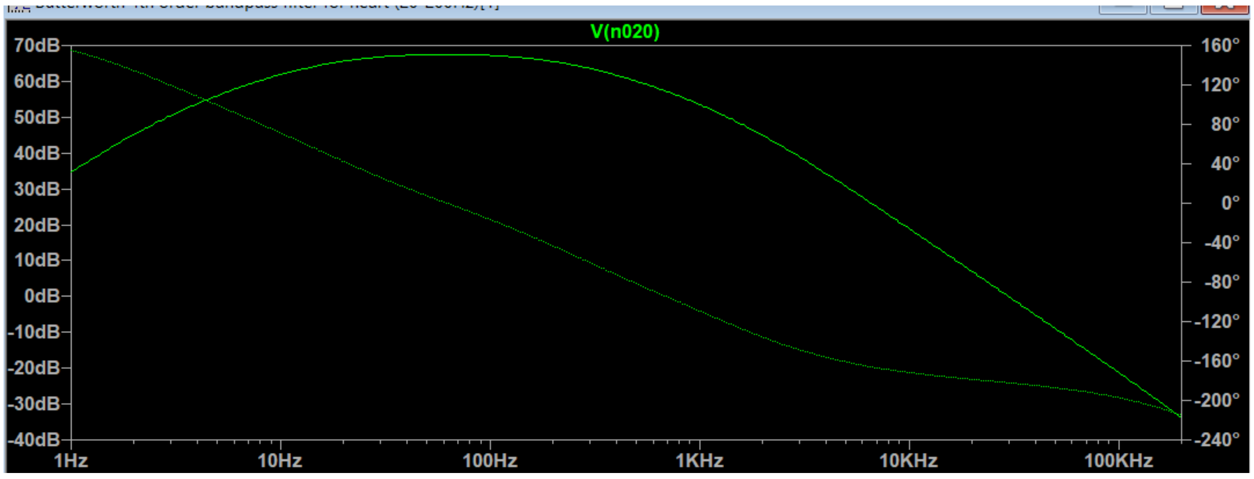


Figure 2.2.2 Second order Bandpass Butterworth Filter (for lungs) Simulation Results

Chapter 3

Signal Analysis

In this chapter, we are analyzing the signal analysis part of the project. Using MATLAB App Designer, we are acquiring the heart and lungs sound signals from the stethoscope circuit into Arduino Uno, and then to the designed app where the analysis section starts.

To be able to analyze signals in MATLAB, we have to write codes that do the work. In this project you will see codes for the following goals.

1. Real Time Signal (Raw) and its spectrum.

2. Filtered Signal and its spectrum.

3. Customizable filter settings for the filtered signal.

Furthermore, we need to develop a GUI design as a user interface. In the GUI of the application, we used two buttons for starting and stopping the analysis - called “Start” and “Stop”. Also, we included a drop-down list to adjust the order of the filter according to the user. The list contains 8 orders of filters (from 1 to 8). Moreover, there is four different plots to display the signals of:

1. Raw Heartbeat signals in time domain.
2. Raw Heartbeat signals in frequency domain (FFT).
3. Filtered Heartbeat signals in Time domain.
4. Filtered Heartbeat signals in frequency domain (FFT).

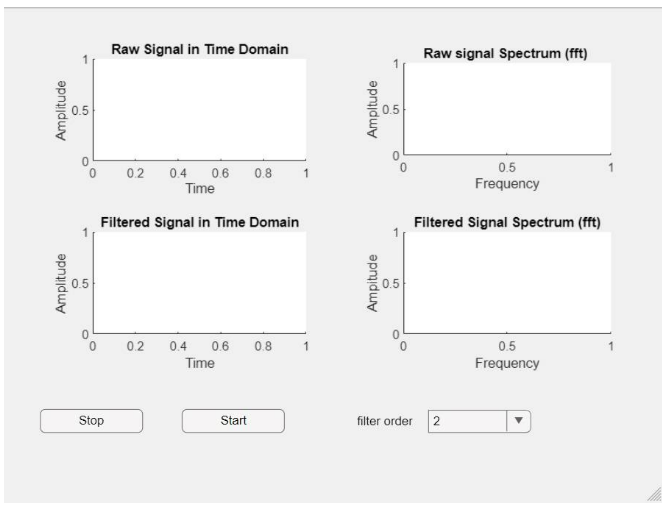
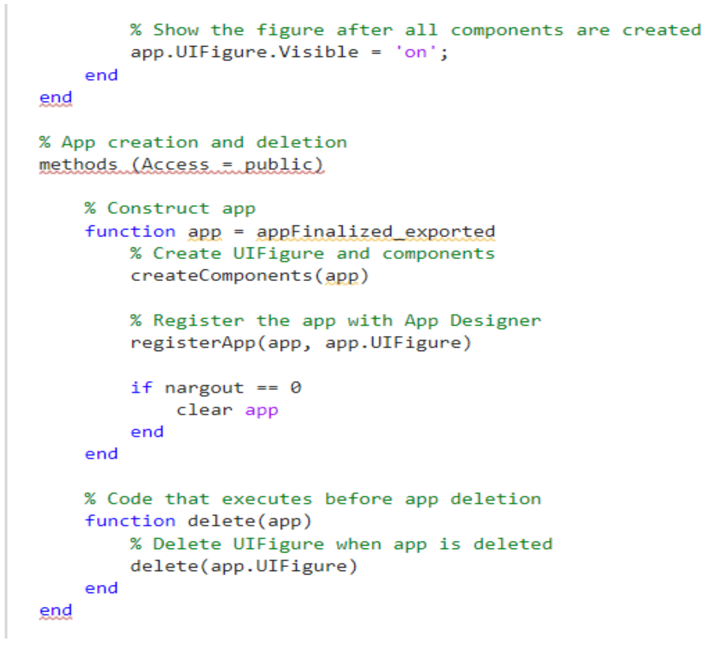
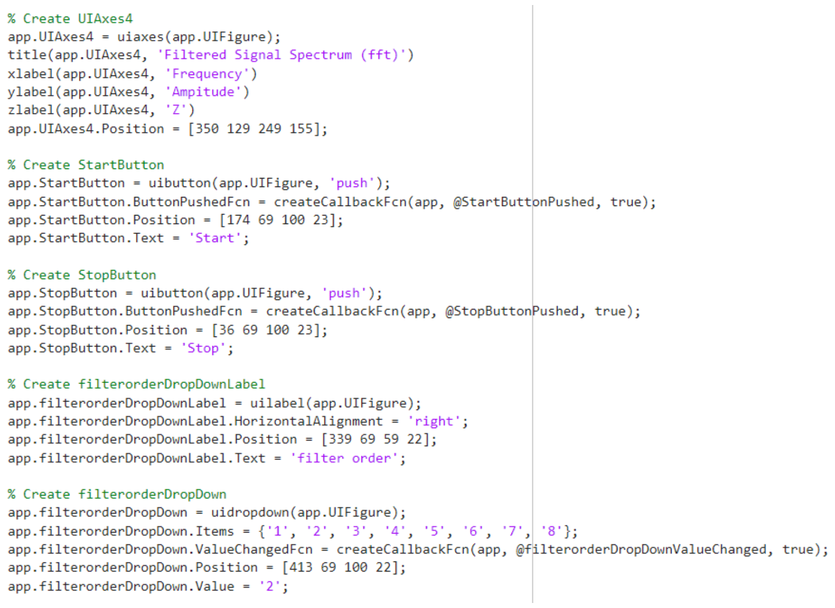

Raw Signal in Time Domain
Amplitude
Time
Raw signal Spectrum (fft)
Amplitude
Frequency
Filtered Signal in Time Domain
Amplitude
Time
Filtered Signal Spectrum (fft)
Amplitude
Frequency
Stop
Start
filter order
2

Figure 3.1.0 (GUI of the application)

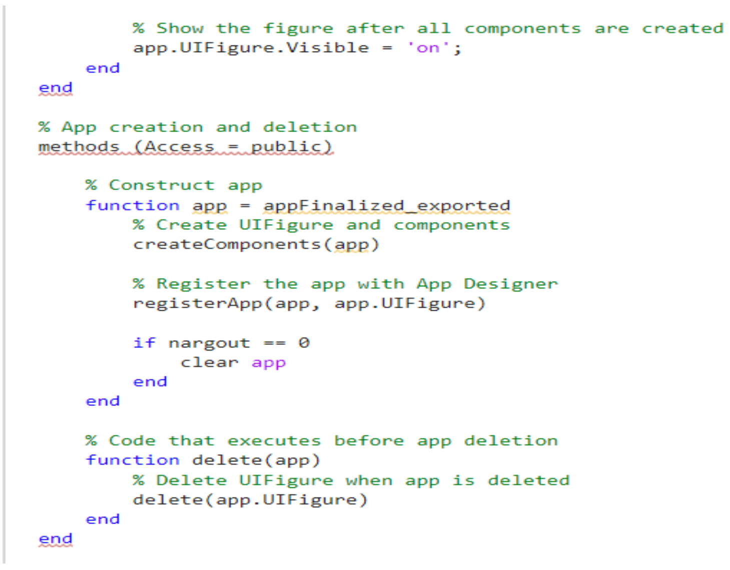
Code for GUI Construction:



```
            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = appFinalized_exported
            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
```

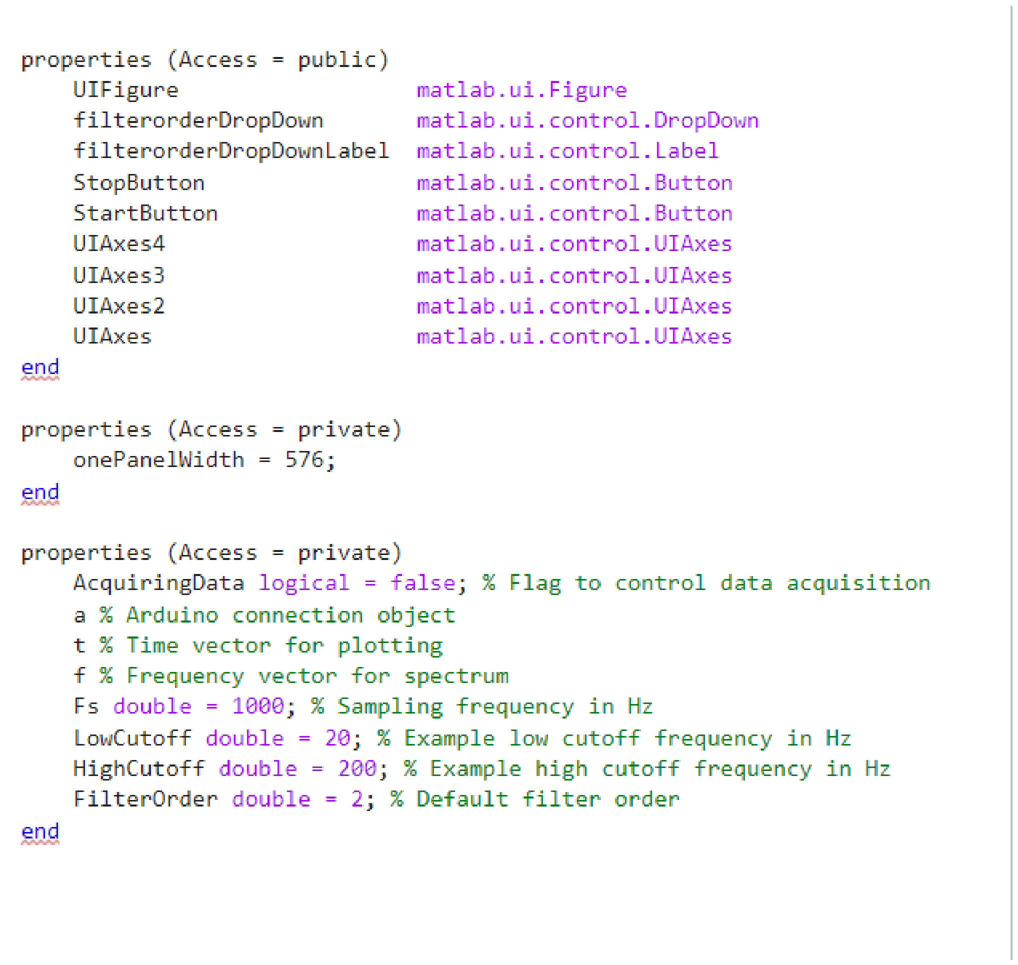


```
% Create UIAxes4
app.UIAxes4 = uiaxes(app.UIFigure);
title(app.UIAxes4, 'Filtered Signal Spectrum (fft)')
xlabel(app.UIAxes4, 'Frequency')
ylabel(app.UIAxes4, 'Ampitude')
zlabel(app.UIAxes4, 'Z')
app.UIAxes4.Position = [350 129 249 155];

% Create StartButton
app.StartButton = uibutton(app.UIFigure, 'push');
app.StartButton.ButtonPushedFcn = createCallbackFcn(app, @StartButtonPushed, true);
app.StartButton.Position = [174 69 100 23];
app.StartButton.Text = 'Start';

% Create StopButton
app.StopButton = uibutton(app.UIFigure, 'push');
app.StopButton.ButtonPushedFcn = createCallbackFcn(app, @StopButtonPushed, true);
app.StopButton.Position = [36 69 100 23];
app.StopButton.Text = 'Stop';

% Create filterorderDropDownLabel
app.filterorderDropDownLabel = uilabel(app.UIFigure);
app.filterorderDropDownLabel.HorizontalAlignment = 'right';
app.filterorderDropDownLabel.Position = [339 69 59 22];
app.filterorderDropDownLabel.Text = 'filter order';

% Create filterorderDropDown
app.filterorderDropDown = uidropdown(app.UIFigure);
app.filterorderDropDown.Items = {'1', '2', '3', '4', '5', '6', '7', '8'};
app.filterorderDropDown.ValueChangedFcn = createCallbackFcn(app, @filterorderDropDownValueChanged, true);
app.filterorderDropDown.Position = [413 69 100 22];
app.filterorderDropDown.Value = '2';
```



```
            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = appFinalized_exported
            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
```

Code for GUI Properties:



```
properties (Access = public)
    UIFigure                 matlab.ui.Figure
    filterorderDropDown      matlab.ui.control.DropDown
    filterorderDropDownLabel matlab.ui.control.Label
    StopButton               matlab.ui.control.Button
    StartButton              matlab.ui.control.Button
    UIAxes4                  matlab.ui.control.UIAxes
    UIAxes3                  matlab.ui.control.UIAxes
    UIAxes2                  matlab.ui.control.UIAxes
    UIAxes                   matlab.ui.control.UIAxes
end

properties (Access = private)
    onePanelWidth = 576;
end

properties (Access = private)
    AcquiringData logical = false; % Flag to control data acquisition
    a % Arduino connection object
    t % Time vector for plotting
    f % Frequency vector for spectrum
    Fs double = 1000; % Sampling frequency in Hz
    LowCutoff double = 20; % Example low cutoff frequency in Hz
    HighCutoff double = 200; % Example high cutoff frequency in Hz
    FilterOrder double = 2; % Default filter order
end
```

Code for GUI Functions:

1. Initialize Arduino Connection:

```matlab
function initializeArduinoConnection(app)
    if isempty(app.a)
        try
            app.a = arduino('COM5', 'Uno', 'Libraries', 'SPI');
            disp('Arduino connected.');
        catch e
            disp('Failed to connect Arduino:');
            disp(e.message);
        end
    end
end
```

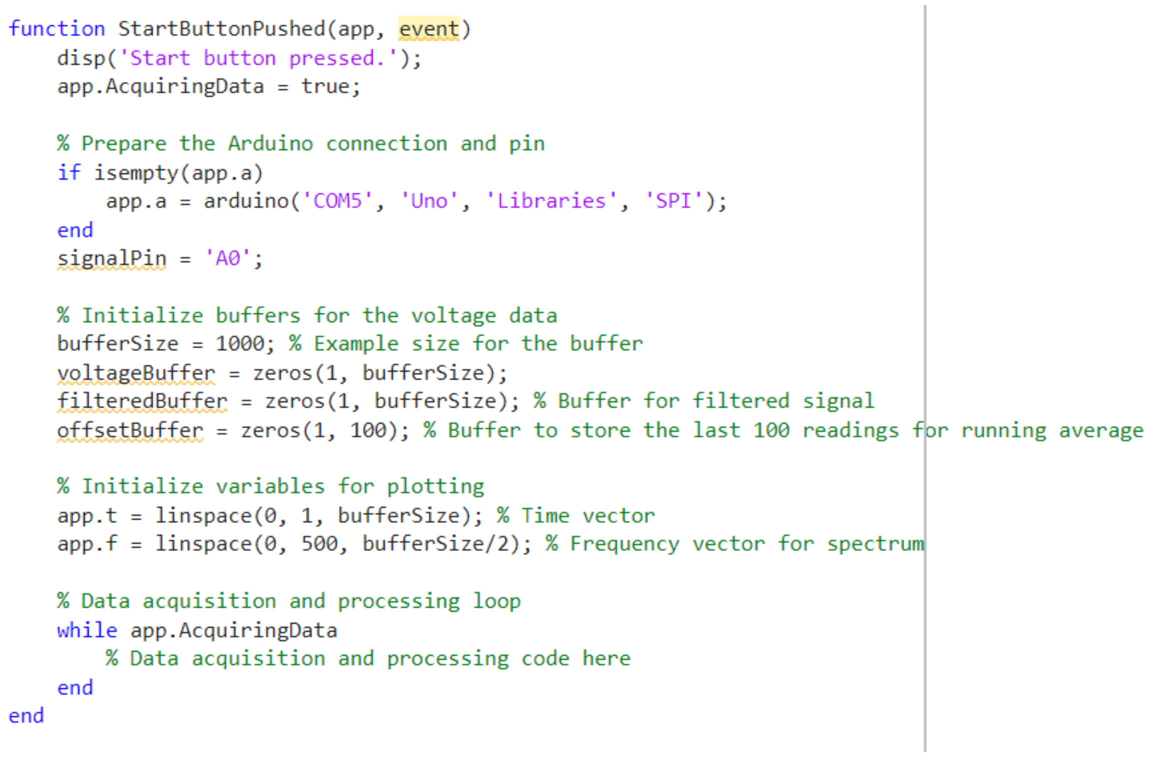
2. Start Function:

```matlab
function startupFcn(app)
    % Initialize Arduino
    initializeArduinoConnection(app);

    % Initialize Time and Frequency Vectors
    app.t = linspace(0, 1, 1000); % Time vector,
    app.f = linspace(0, 500, 500); % Frequency vector for spectrum,
end
```

3. Update Plots:

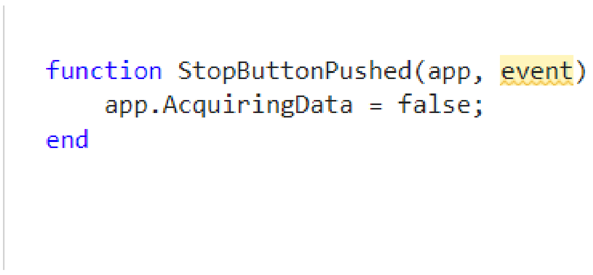
```
function updatePlots(app, rawData)
    % Plot the raw signal using stairs
    stairs(app.UIAxes, app.t, rawData);
    title(app.UIAxes, 'Raw Signal in Time Domain');
    ylim(app.UIAxes, 'auto');

    % Calculate and plot the spectrum of the raw signal
    Y = fft(rawData);
    P2 = abs(Y/length(rawData));
    P1 = P2(1:length(rawData)/2+1);
    P1(2:end-1) = 2*P1(2:end-1);
    plot(app.UIAxes3, app.f, P1);
    title(app.UIAxes3, 'Spectrum of Raw Signal (fft)');
end
```

4. Start Button Pushed Function:



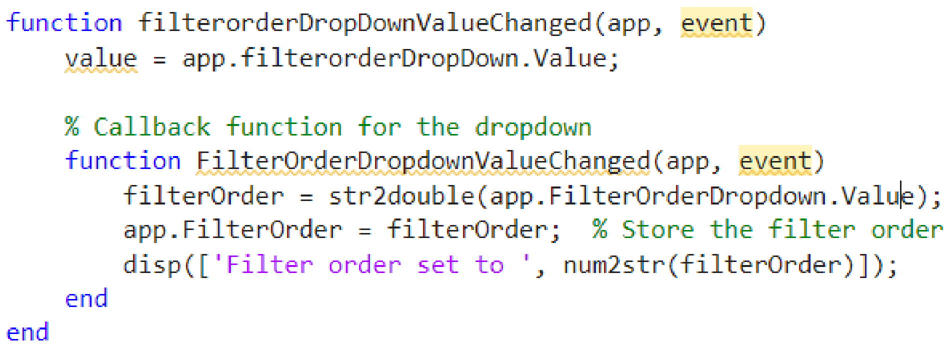
```
function StartButtonPushed(app, event)
    disp('Start button pressed.');
    app.AcquiringData = true;

    % Prepare the Arduino connection and pin
    if isempty(app.a)
        app.a = arduino('COM5', 'Uno', 'Libraries', 'SPI');
    end
    signalPin = 'A0';

    % Initialize buffers for the voltage data
    bufferSize = 1000; % Example size for the buffer
    voltageBuffer = zeros(1, bufferSize);
    filteredBuffer = zeros(1, bufferSize); % Buffer for filtered signal
    offsetBuffer = zeros(1, 100); % Buffer to store the last 100 readings for running average

    % Initialize variables for plotting
    app.t = linspace(0, 1, bufferSize); % Time vector
    app.f = linspace(0, 500, bufferSize/2); % Frequency vector for spectrum

    % Data acquisition and processing loop
    while app.AcquiringData
        % Data acquisition and processing code here
    end
end
```

5. Stop Button Pushed Function:



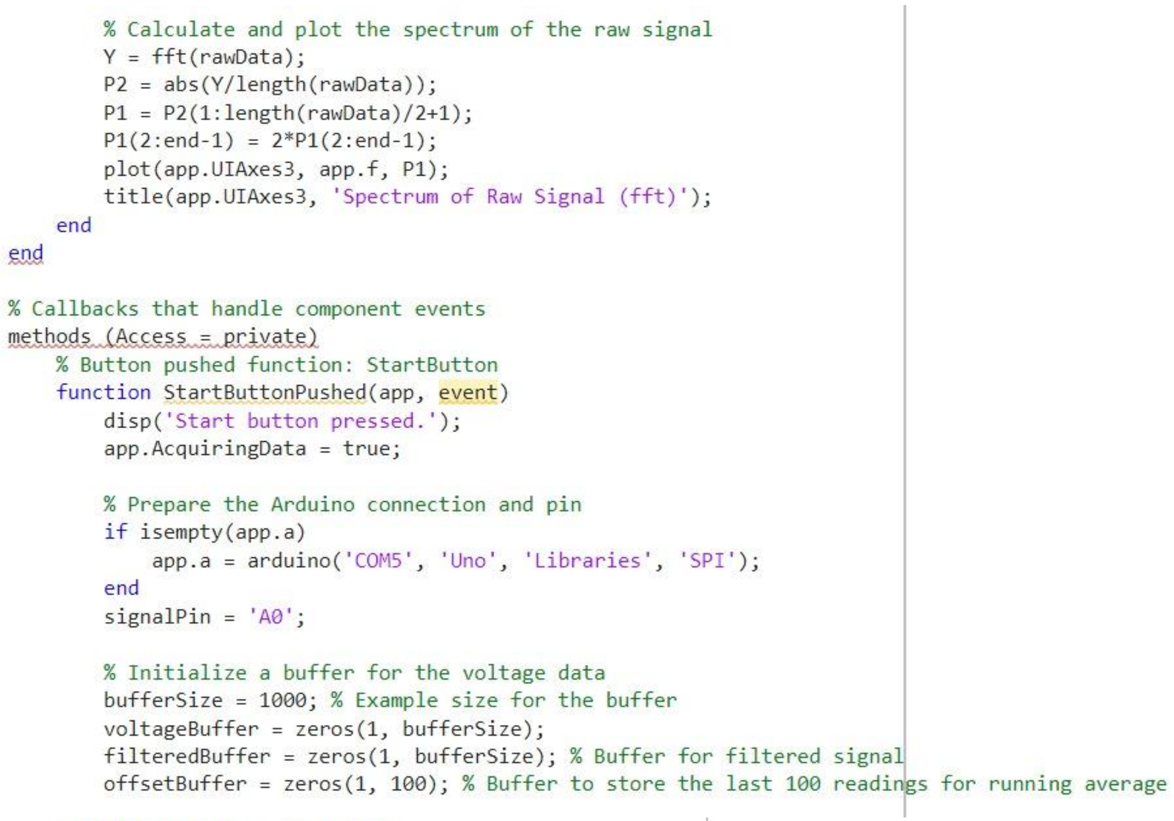
```
function StopButtonPushed(app, event)
    app.AcquiringData = false;
end
```

6. Filter Order Dropdown Value Changed Function

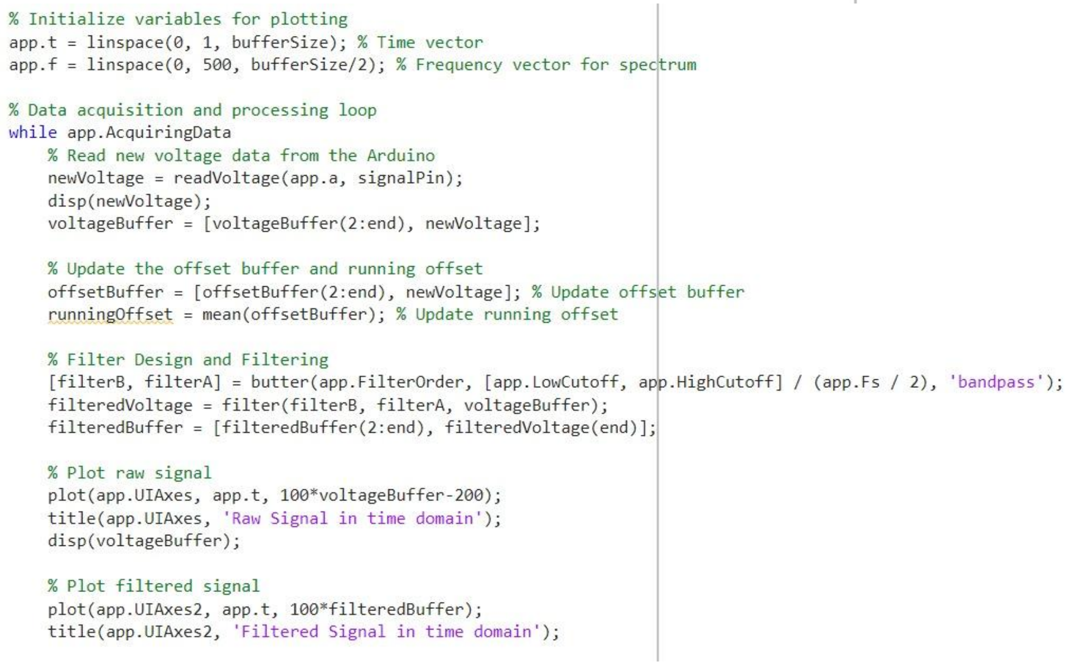


```
function filterorderDropDownValueChanged(app, event)
    value = app.filterorderDropDown.Value;

    % Callback function for the dropdown
    function FilterOrderDropdownValueChanged(app, event)
        filterOrder = str2double(app.FilterOrderDropdown.Value);
        app.FilterOrder = filterOrder;  % Store the filter order
        disp(['Filter order set to ', num2str(filterOrder)]);
    end
end
```

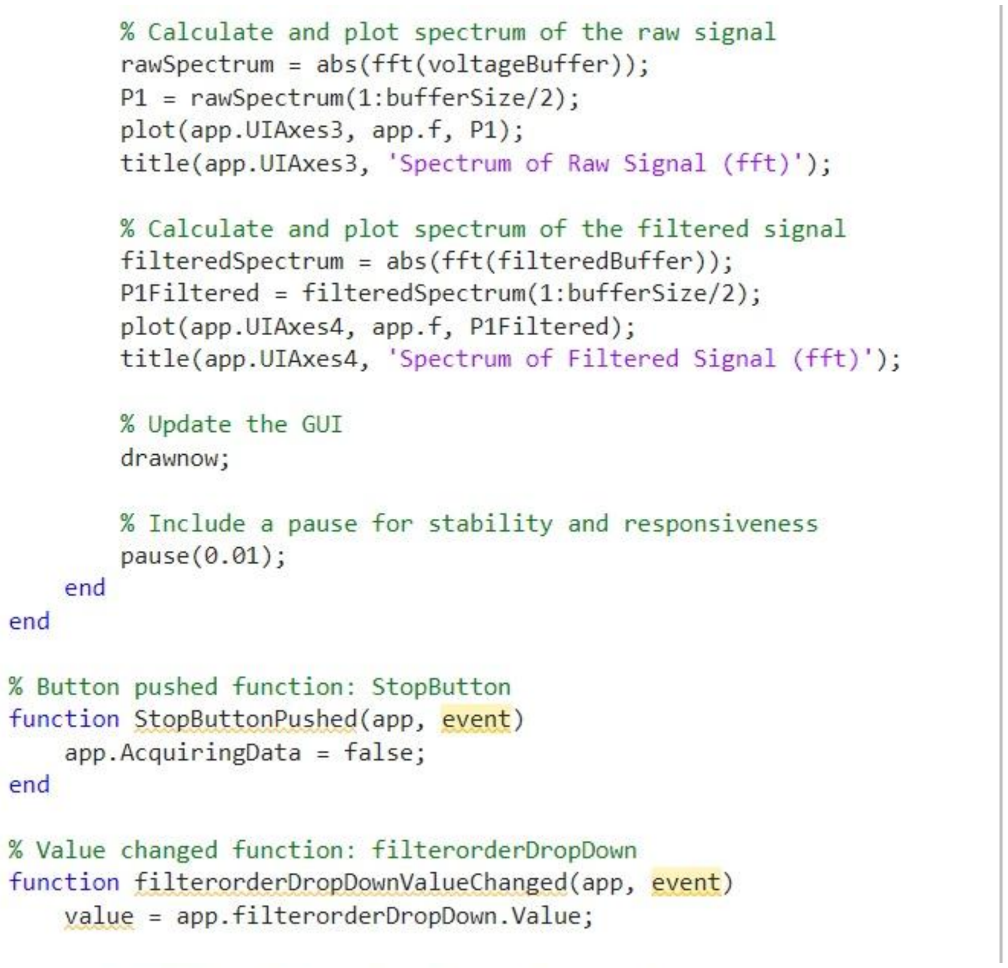
Code for Methods



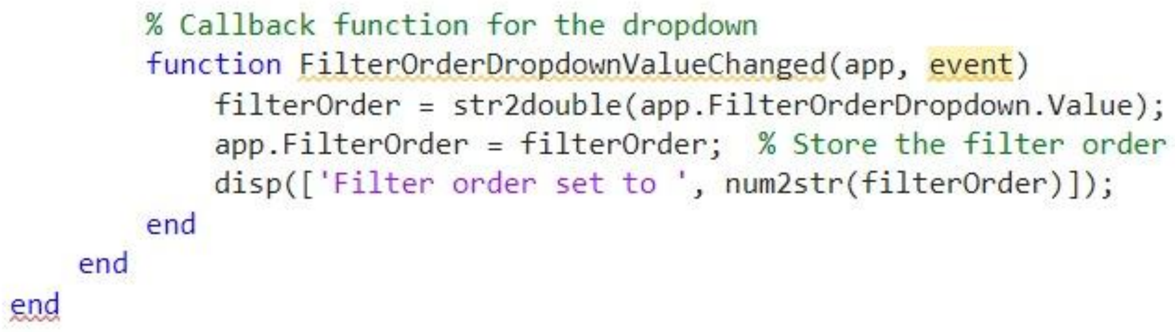
```matlab
            % Calculate and plot the spectrum of the raw signal
            Y = fft(rawData);
            P2 = abs(Y/length(rawData));
            P1 = P2(1:length(rawData)/2+1);
            P1(2:end-1) = 2*P1(2:end-1);
            plot(app.UIAxes3, app.f, P1);
            title(app.UIAxes3, 'Spectrum of Raw Signal (fft)');
        end
    end

    % Callbacks that handle component events
    methods (Access = private)
        % Button pushed function: StartButton
        function StartButtonPushed(app, event)
            disp('Start button pressed.');
            app.AcquiringData = true;

            % Prepare the Arduino connection and pin
            if isempty(app.a)
                app.a = arduino('COM5', 'Uno', 'Libraries', 'SPI');
            end
            signalPin = 'A0';

            % Initialize a buffer for the voltage data
            bufferSize = 1000; % Example size for the buffer
            voltageBuffer = zeros(1, bufferSize);
            filteredBuffer = zeros(1, bufferSize); % Buffer for filtered signal
            offsetBuffer = zeros(1, 100); % Buffer to store the last 100 readings for running average
```



```matlab
            % Initialize variables for plotting
            app.t = linspace(0, 1, bufferSize); % Time vector
            app.f = linspace(0, 500, bufferSize/2); % Frequency vector for spectrum

            % Data acquisition and processing loop
            while app.AcquiringData
                % Read new voltage data from the Arduino
                newVoltage = readVoltage(app.a, signalPin);
                disp(newVoltage);
                voltageBuffer = [voltageBuffer(2:end), newVoltage];

                % Update the offset buffer and running offset
                offsetBuffer = [offsetBuffer(2:end), newVoltage]; % Update offset buffer
                runningOffset = mean(offsetBuffer); % Update running offset

                % Filter Design and Filtering
                [filterB, filterA] = butter(app.FilterOrder, [app.LowCutoff, app.HighCutoff] / (app.Fs / 2), 'bandpass');
                filteredVoltage = filter(filterB, filterA, voltageBuffer);
                filteredBuffer = [filteredBuffer(2:end), filteredVoltage(end)];

                % Plot raw signal
                plot(app.UIAxes, app.t, 100*voltageBuffer-200);
                title(app.UIAxes, 'Raw Signal in time domain');
                disp(voltageBuffer);

                % Plot filtered signal
                plot(app.UIAxes2, app.t, 100*filteredBuffer);
                title(app.UIAxes2, 'Filtered Signal in time domain');
```



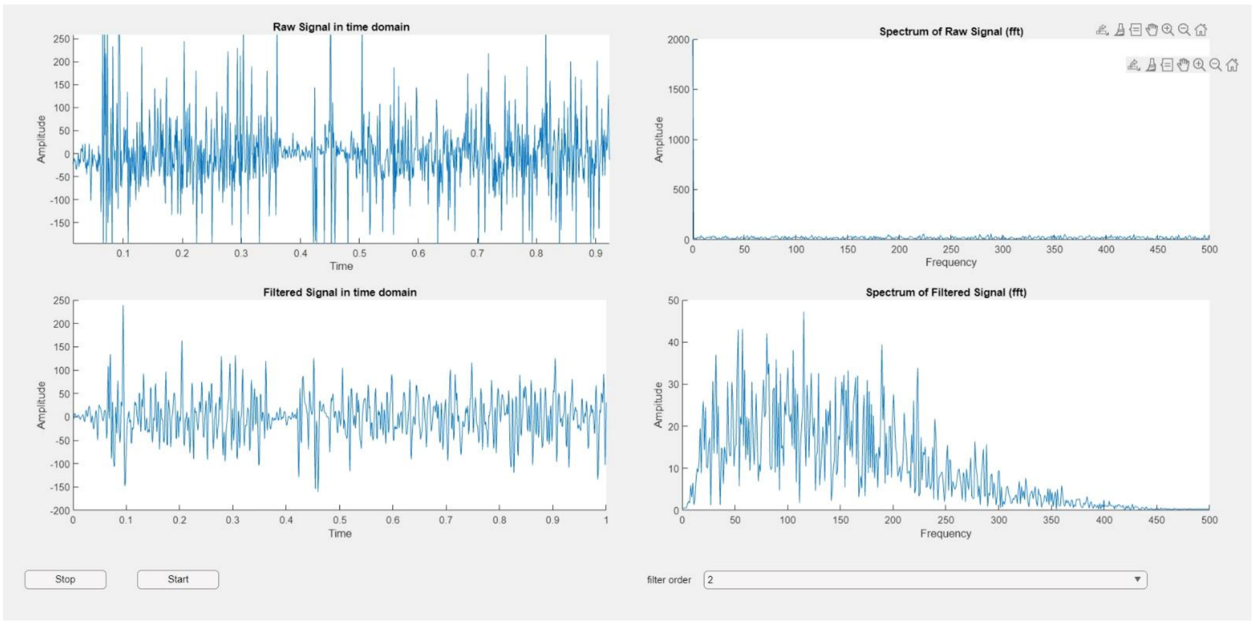
```
                % Calculate and plot spectrum of the raw signal
                rawSpectrum = abs(fft(voltageBuffer));
                P1 = rawSpectrum(1:bufferSize/2);
                plot(app.UIAxes3, app.f, P1);
                title(app.UIAxes3, 'Spectrum of Raw Signal (fft)');

                % Calculate and plot spectrum of the filtered signal
                filteredSpectrum = abs(fft(filteredBuffer));
                P1Filtered = filteredSpectrum(1:bufferSize/2);
                plot(app.UIAxes4, app.f, P1Filtered);
                title(app.UIAxes4, 'Spectrum of Filtered Signal (fft)');

                % Update the GUI
                drawnow;

                % Include a pause for stability and responsiveness
                pause(0.01);
            end
        end

        % Button pushed function: StopButton
        function StopButtonPushed(app, event)
            app.AcquiringData = false;
        end

        % Value changed function: filterorderDropDown
        function filterorderDropDownValueChanged(app, event)
            value = app.filterorderDropDown.Value;
```



```
        % Callback function for the dropdown
        function FilterOrderDropdownValueChanged(app, event)
            filterOrder = str2double(app.FilterOrderDropdown.Value);
            app.FilterOrder = filterOrder;  % Store the filter order
            disp(['Filter order set to ', num2str(filterOrder)]);
        end
    end
end
```

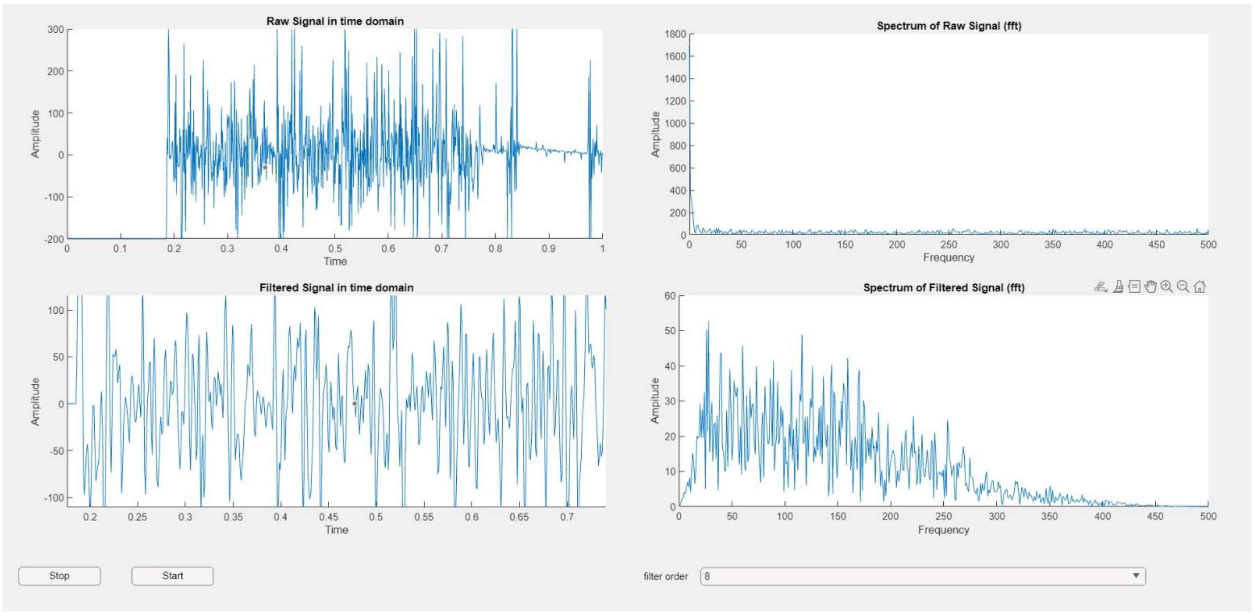
Now, after explaining the code section of our project app, let us see the output results of our code.

1. Heart Analysis:


Raw Signal in time domain
Spectrum of Raw Signal (fft)
Filtered Signal in time domain
Spectrum of Filtered Signal (fft)
Amplitude
Time
Frequency
Stop
Start

filter order 2

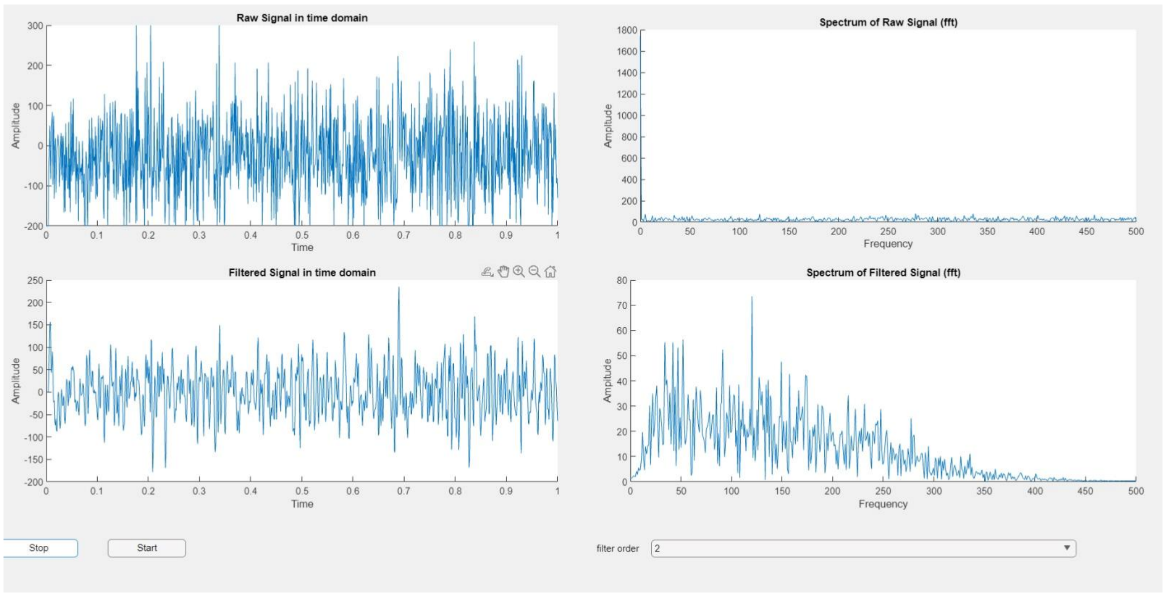
Figure 3.2.0 (Second Order Heart Beat analysis).


Raw Signal in time domain
Spectrum of Raw Signal (fft)
Filtered Signal in time domain
Spectrum of Filtered Signal (fft)
Amplitude
Time
Frequency
Stop
Start

filter order 8

Figure 3.3.0 (8th Order Heart Beat analysis).

2. Lung analysis:


Raw Signal in time domain
Amplitude
Time
Spectrum of Raw Signal (fft)
Frequency
Filtered Signal in time domain
Spectrum of Filtered Signal (fft)
Stop
Start
filter order
2

For further information about the app, check out the GitHub link below:

https://github.com/FurqanBhat/Electronic-Stethoscope-for-heart-beat-and-respiration-system.git

Conclusion

For this project, we created and assembled an electronic stethoscope circuit with a finely adjusted filter to accurately record and examine the varied sounds produced by different body parts. The circuit gave us the opportunity to investigate the complex acoustics of the human body as we saw the data in real-time. After the data was gathered, Matlab was used to analyze it, allowing us to see and understand the minute details in the physiological sounds. Our project included the installation and testing of a basic Electrocardiogram (ECG) device in addition to the stethoscope capability. This section turned out to be very important for the medical field since it made it easier to capture and examine the electrical activity of the heart. The project's comprehensive nature not only enhanced our comprehension of electrical circuits and signal processing, but also highlighted the usefulness of this technology in improving healthcare professionals' diagnostic tools. Our practical experience allowed us to acquire important insights into the relationship between electronics and medical instruments, which expanded our skill set in the engineering and healthcare areas.

References

1) Electrical4U. (2021). [Butterworth Filter: What is it? (Design & Applications)]. Retrieved from https://www.electrical4u.com/butterworth-filter/

2) HTM workshop. (2022). [Can you build your own Electronic Stethoscope? Learn how to build and amplifier circuit]. Retrieved from https://www.youtube.com/watch?v=w6MbM_f_7Uo

3) Leng, Tan,& Ghista. (2015). [The electronic stethoscope]. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4496820/

4) Wu, M., & Deng, G. (2012). [Digital Stethoscope]. ECE 4760: Designing with Microcontrollers, Cornell University. Retrieved from https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/myw9_gdd9/myw9_gdd9/